3E-6

C－Prolog コンパイラの性能評価

碇崎　賢一　上原　邦昭　豊田　順一

（大阪大学産業科学研究所）

1　はじめに

人工知能の研究において PROLOG が広く用いられている。PROLOG は基本的にインタプリタによって実行される言語であるが、大規模なシステムや実用的なシステムを構築するには、コンパイラによる処理の高速化が必須である。

筆者らは PROLOG を高速に実行するためのコンパイラの実現手法に関する研究を行っており、実用的な処理系としてスーパーミニコン MV/8000Ⅱ上に C-Prolog コンパイラを開発している。テールリカージョンの最適化等を行った結果、25 K LIPS の処理性能を得ている。本稿では、C-Prolog コンパイラで実現している PROLOG の特性に適した使用方法と、性能評価について報告する。

2　C-Prolog コンパイラの特徴

C-Prolog コンパイラはターゲットマシンの機械語を生成するインコアコンパイラで、C-Prolog インタプリタとともに稼動する。処理系の開発においては、PROLOG プログラムを高速に処理するためのメカニズムを研究するだけでなく、PROLOG のプロトタイピング言語としての特徴を十分に生かすために２つの機能を実現している。１つめはコンパイルされた述語とされていない述語を同一環境で混在して実行するための機能で、２つめは簡便に利用できるインクリメンタルなコンパイル機能である。この他の特徴として、C-Prolog インタプリタが提供している環境をすべて利用できること、システムのほとんどの部分を PROLOG で記述しているため保守しやすくなっていることなどがある。

3　コンパイラの使用法

PROLOG の特徴を生かすため、コンパイラの利用形態はインクリメンタルなインコアコンパイラ方式としている。この方式では、プログラム全体をコンパイルする必要はなく、インタプリタによるテストが終わった述語から順にコンパイルして利用することができる。また、コンパイルのために現在の処理を中断したり、リンカを使って毎回実行ファイルを作成する必要がない。

プログラムのコンパイル法は、述語単位でコンパイルする方法と、関連する述語群を１つのモジュールとしてコンパイルする方法の２種類を実現している。モジュールとしてコンパイルされた述語群は密に結合されるため、述語単位でコンパイルされた述語と比較すると、ゴールの呼び出しのオーバーヘッドがないという利点がある。また、コンパイラが生成したコードはリロケータブルコードとして保存でる。

述語単位のコンパイルは compile 述語を用いて、コンパイルする述語とそのアリティを指定して行う。

```
:- compile(pred(n)).
```

ここで pred は述語名 n はアリティである。

モジュール単位のコンパイルは mcompile 述語を用いて、コンパイルする述語とそのアリティをリスト形式で指定して行う。

```
:- mcompile([pred1(n1),pred2(n2),...]).
```

コンパイルコードをリロケータブルファイルとして保存する場合には、述語のリストと共に出力ファイル名を指定する。

```
:- mcompile([pred1(n1),...],file).
```

リロケータブルコードのローディングは mload 述語を用いて、リロケータブルファイル名を指定して行う。

```
:- mload(file).
```

4　コード生成と最適化手法

4.1 コンパイルコードの生成

コンパイルコードの生成法は、ソースプログラムをC言語等のコンパイル言語に変換する方法や中間言語に変換する方法などが考えられる。しかしながら、本コンパイラでは高速性を追求するために、ターゲットマシンの機械語を生成する方式をとっている。実際の処理では、ソースプログラムを直接機械語に変換するのではなく、まず仮想 PROLOG マシン(PLM)[1]の命令である PLM コードに変換し、次にこのコードをターゲットマシンであるスーパーミニコン MV/8000Ⅱの機械語に変換する２段階のコンパイル方式をとっている。

4.2 PROLOG における最適化手法

PROLOG 処理系での基本的な最適化手法として、1) モード宣言の利用、2) インデッキシング、3) テールリカージョンの最適化の３つがあげられる。モード宣言は単独で使用されるだけでなく、インデッキシングやテールリカージョンの最適化に利用される。ここでは特に，2), 3)について説明する。

4.2.1 インデッキシング

モード宣言で第１引数が入力として宣言された場合は、第１引数の値をキーとしてインデッキシングを行う。インデッキシングの処理は２段階に分けられる。１段めはアトムや複合項といった入力項のタイプによるインデッ

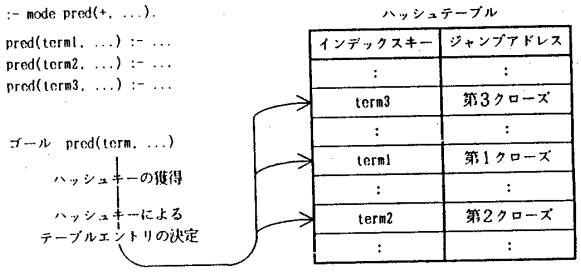

図1　ハッシュ法を用いたインデッキシング

Performance and Evaluation of C-Prolog Compiler

Ken'ichi KAKIZAKI, Kuniaki UEHARA, Jyun'ichi TOYODA

Institute of Scientific and Industrial Research, OSAKA Univ.

キシング、2段めは1段めで分類された後の同一タイプ内でのインデッキシングである。

2段めのインデッキシングを行う手法として、ハッシュ法が提案されている。ハッシュ法はインデッキシングのキーとなる値からハッシュキーを生成し、このキーを用いて実行すべきクローズを決定する（図1）ものである。インデッキシングのキーとして用いるのは、第1引数として入力された値である。これは、入力項のアトムあるいは複合項のファンクタに関する情報を格納している領域を指すポインタとなっている。

本コンパイラはインタプリタと共存しているため、アトムやファンクタの登録と削除を自由に行うことができる。アトムやファンクタはメモリ内で動的に管理されているため、システムの起動ごとにこれらを指すポインタの値が変わる可能性がある。ところが、インデッキシングに使用するハッシュ法は、テーブルを参照するためにいつでも同一の項から同一のキーを生成する必要がある。したがって、同一の項に対して値が変わる可能性があるポインタを、ハッシュキーを生成する基として使用することはできない。この問題を解決するために採用した方法を次に示す。

インタプリタはプログラムの読み込み時にアトムやファンクタに遭遇すると、それらを内部のデータベースに対して高速に照会および登録を行うために、その文字列からハッシュキーを生成する。このハッシュキーは、それぞれのアトムやファンクタに固有の値となるため、インデッキシングに利用することができる。本コンパイラでは、このハッシュキーをインデッキシングに利用するため、インタプリタの読み込み機構に機能の追加を行い、読み込み時に生成したハッシュキーを、アトムあるいはファンクタのそれぞれのデータベースに格納するようにしている。コンパイラは、データベースに既に格納されているハッシュキーを参照して、インデッキシングで使用するジャンプテーブルを構成する。

インタプリタの読み込み機構で使用するハッシュサイズは 269 で、生成されるハッシュキーは 0〜268 の値をとる。ハッシュ法ではハッシュテーブルのエントリ数をハッシュサイズと同じ大きさにとるのが普通である。しかし、クローズ数が少ないインデッキシングでは用意したエントリーのほとんどが無駄になってしまう。そこでクローズ数の少ない述語のインデッキシングでは、ハッシュキーの下位数ビットのみを使用してインデッキシングを行い、メモリの浪費を防止している。例えば、10個のクローズがある場合は、ハッシュキーの下位4ビットを使用し 16 エントリーのジャンプテーブルを使ってインデッキシングを行っている。

4.2.2 テールリカージョンの最適化

本コンパイラはテールリカージョンの最適化（TRO）を行うために、引数レジスタをもうけ、TRO を行ったコードがこのレジスタを用いて引数の受け渡しを行うようにしている。また、引数レジスタを使用したユニフィケーション命令やコール命令等を PLM コードレベルで拡張している。これらのレジスタと命令群を使用することで TRO を効率よく処理している。

5 性能評価

ベンチマークテストには、PROLOG コンテスト[4]の課題として用いられた nreverse と qsort を使用した。1回あたりの処理量は、それぞれ 496 LI と 609 LI である。実行時間の計測にはＣＰＵの内部クロックを用い、10 回ループさせた時の実行時間を測定した。プログラムの1回あたりの実行時間と LIPS 値、およびインタプリタとコンパイルコードとの速度比を図2に示す。

	実行時間(ms)	性能(LIPS)	速度比
ｲﾝﾀﾌﾟﾘﾀ	280.0	1.77K	1
ｺﾝﾊﾟｲﾙｺｰﾄﾞ-1	53.3	9.31K	5.3
ｺﾝﾊﾟｲﾙｺｰﾄﾞ-2	20.0	24.80K	14.0

1） nreverse のベンチマーク

	実行時間(ms)	性能(LIPS)	速度比
ｲﾝﾀﾌﾟﾘﾀ	430.0	1.42K	1
ｺﾝﾊﾟｲﾙｺｰﾄﾞ-1	68.3	8.91K	6.3
ｺﾝﾊﾟｲﾙｺｰﾄﾞ-2	26.5	23.00K	16.2

2） qsort のベンチマーク

ｺﾝﾊﾟｲﾙｺｰﾄﾞ-1　最適化を行っていないコード
ｺﾝﾊﾟｲﾙｺｰﾄﾞ-2　最適化を行ったコード

図2　ベンチマーク結果

最適化手法を利用して生成したコードは、nreverse, qsort ともに、それぞれ 25K, 23K LIPS と同レベルの汎用プロセッサ上の処理系としては、高い性能を示している。この性能をインタプリタと比較すると、それぞれ14倍、16倍と高速化されており、コンパイラ作成の意義が認められる。

PROLOG プログラムの実行は、TRO によって処理速度は2倍程度の高速化が行われる[2]といわれている。最適化手法を利用して生成したコードは、そうでないコードに比べ 2.6 倍程度高速化されている。この高速化の要因として、最適化されたコードでは環境の保存や再設定が必要最小限しかおこなわれないこと、neck, foot 処理が省かれること、あるいはユニフィケーション処理が最適化されていることなどが考えられる。

6 現状と今後の予定

現在も引き続きコンパイラを改良中で、特に2段めのインデッキシングを実現する作業に重点を置いて進めている。また、研究室に新たに導入される SUN-3 ワークステーション上にコンパイラを移植することを検討中である。SUN-3 は MV/8000Ⅱの2倍程度の処理能力を持っており、本コンパイラを移植することにより 50K LIPS 程度の高い処理速度を達成できると考えられる。

参考文献

[1] Warren, D.H.D.: Implementing Prolog: compiling predicate logic programs, Dept. of Artificial Intelligence, University of Edinburgh, Research Reports 39 & 40, 1977.

[2] Warren, D.H.D.: An Improved Prolog Implementation which Optimises Tail Recursion, Dept. of Artificial intelligence, University of Edinburgh, Research Paper No.141, 1980.

[3] 砲崎, 上原, 豊田： C-Prolog コンパイラの開発, ICOT, Proceedings of the Logic Programming Conference '86, pp.159-166, 1986.

[4] 奥乃： 第三回 LISP コンテストおよび第一回 PROLOG コンテストの課題案, 情報処理学会, 記号処理研究会資料, 28-4, 1984.